OPERATING SYSTEMS

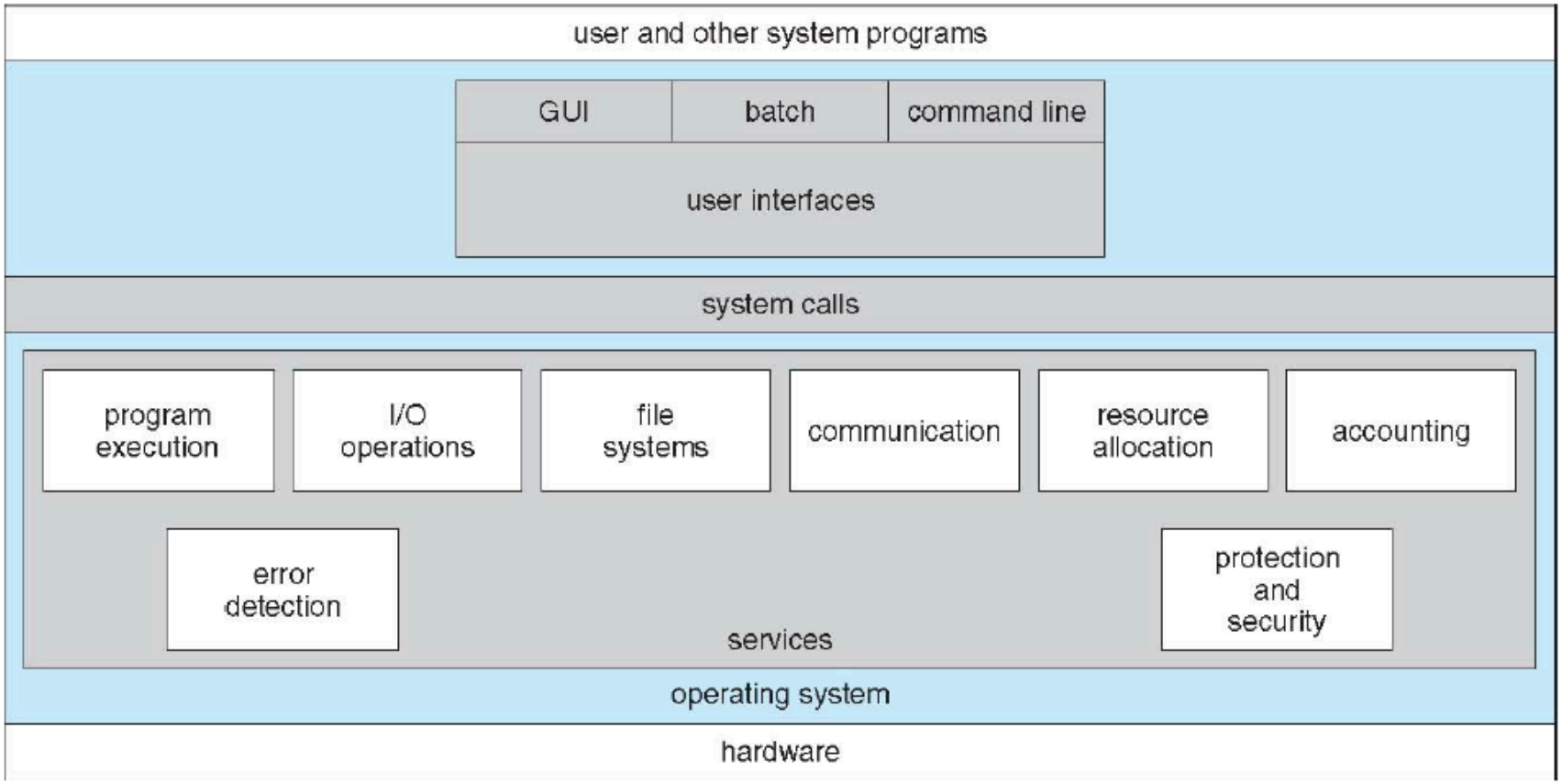
Lesson 2: OS STRUCTURES

Nguyễn Thị Hậu (UET-VNU)
nguyenhau@vnu.edu.vn

CONTENTS

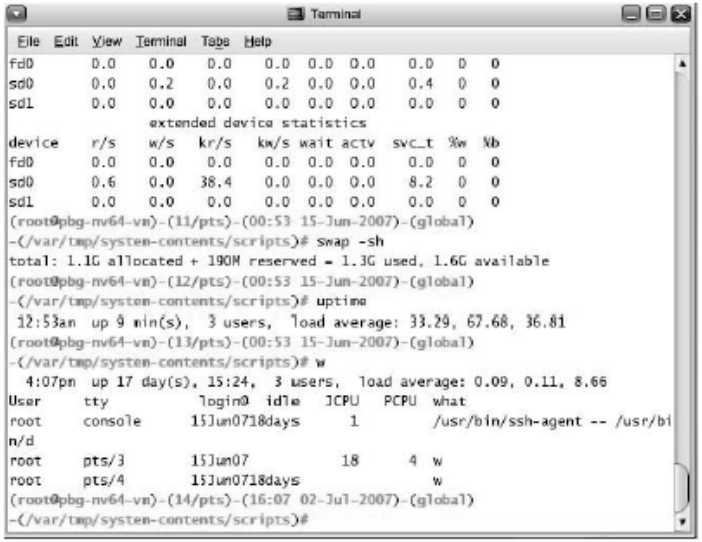
- Operating System Services
- User and Operating System-Interface
- System Calls
- System Services
- Operating System Structure

Operating System Services



User Operating System Interface

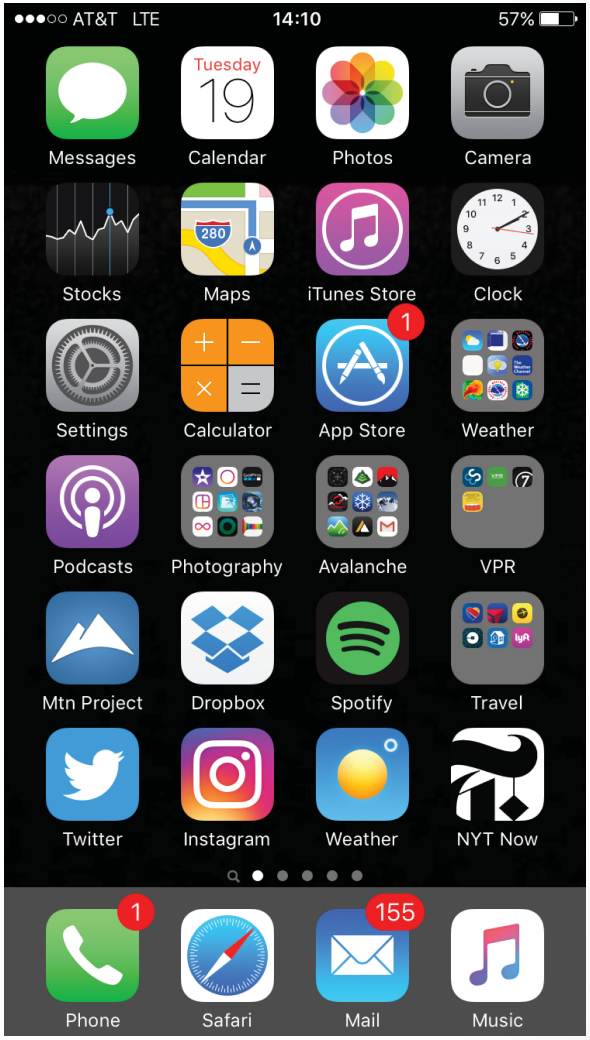
- Command Interpreter - CI



- Graphical User Interface - GUI
  - Aero : Window Vista/Window 7/8/10
  - Aqua: MacOS X
  - Common desktop environment (CDE): Unix, Linux OpenVMS
  - K desktop environment (KDE): Unix
  - GNOME desktop (GNOME) : Ubuntu

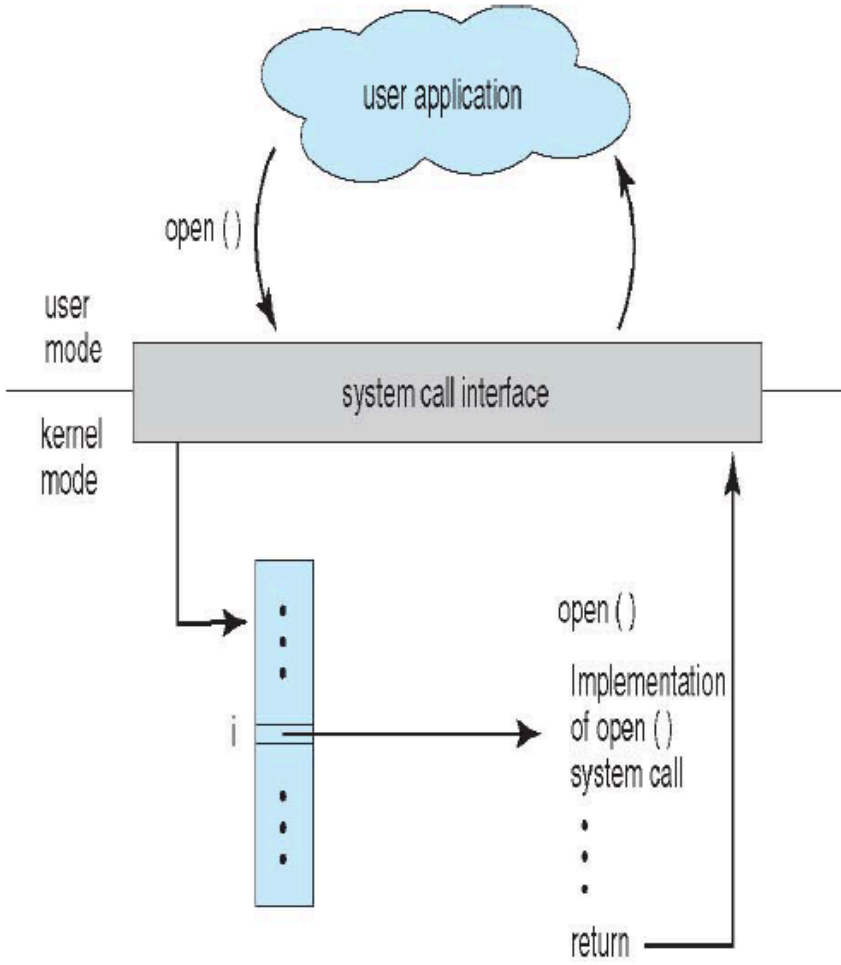
Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands



System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)



Examples of Windows, Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

An example of standard API

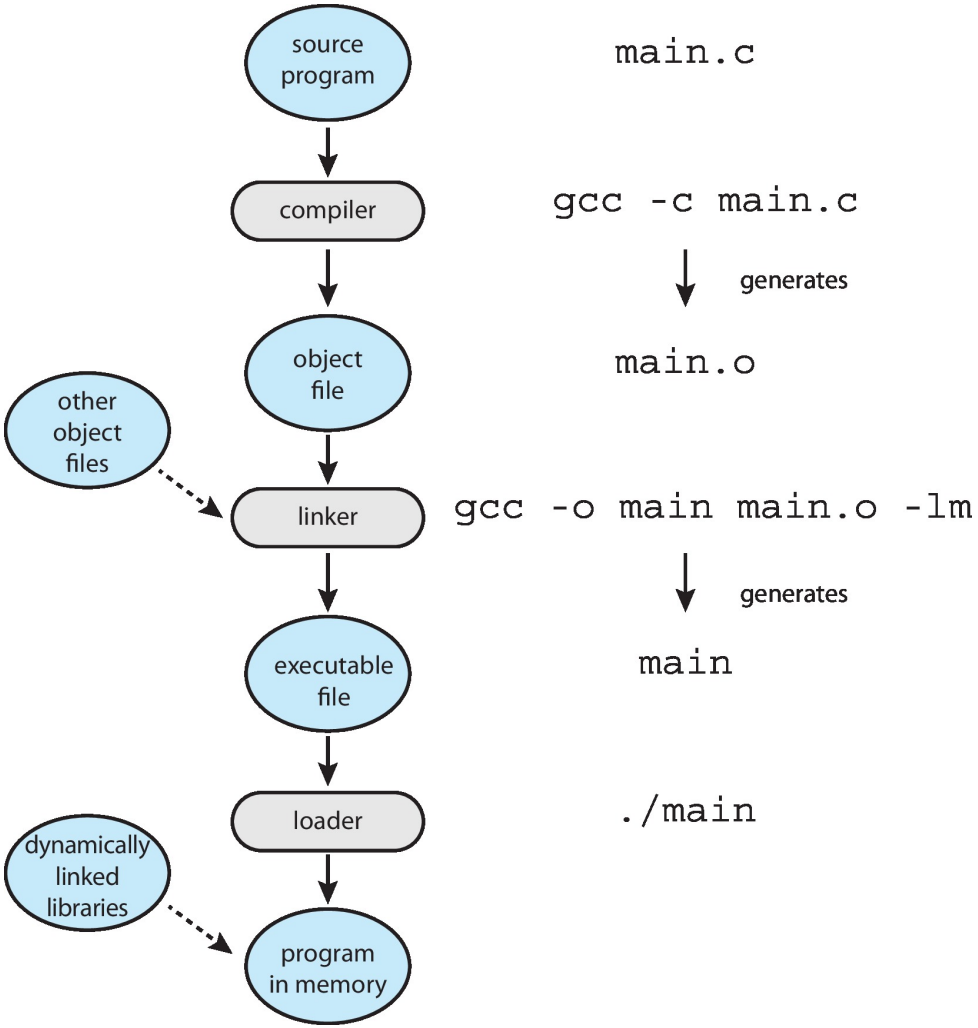
```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value | function name | parameters

System Services

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a file
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

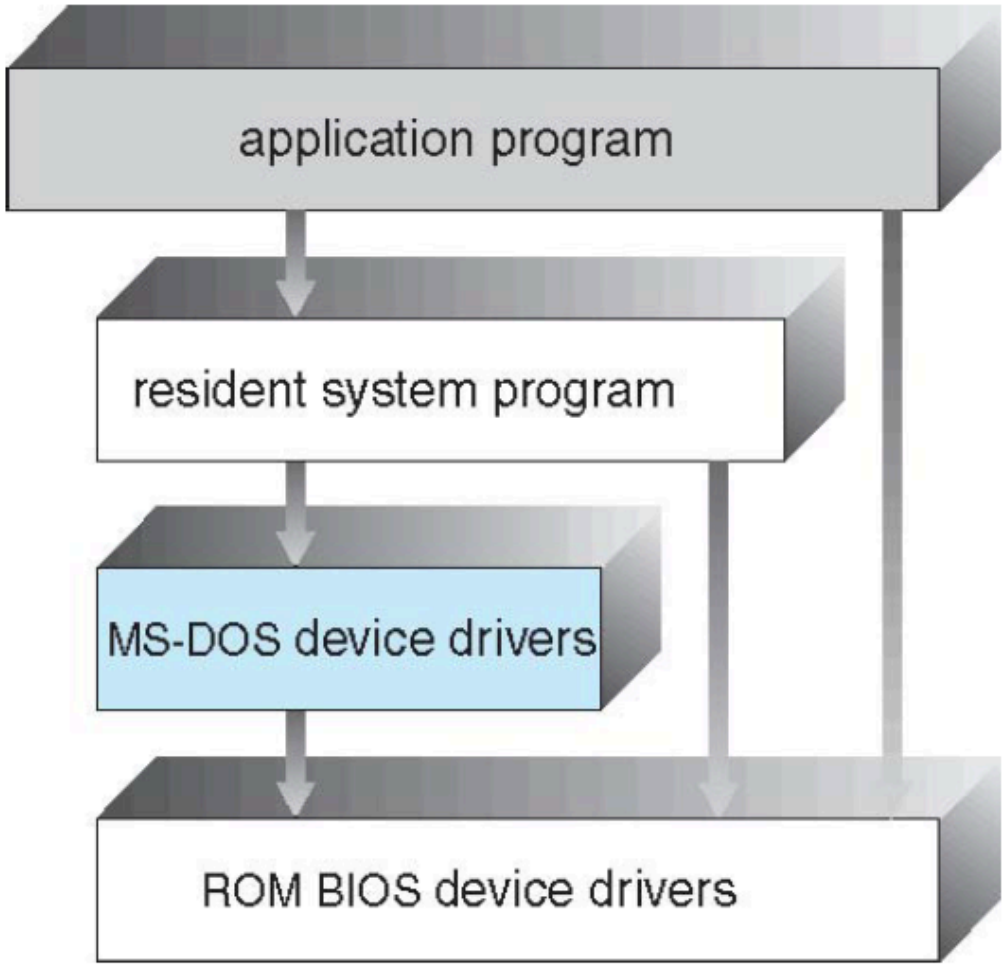
The Role of the Linker and Loader



OPERATING SYSTEM STRUCTURE

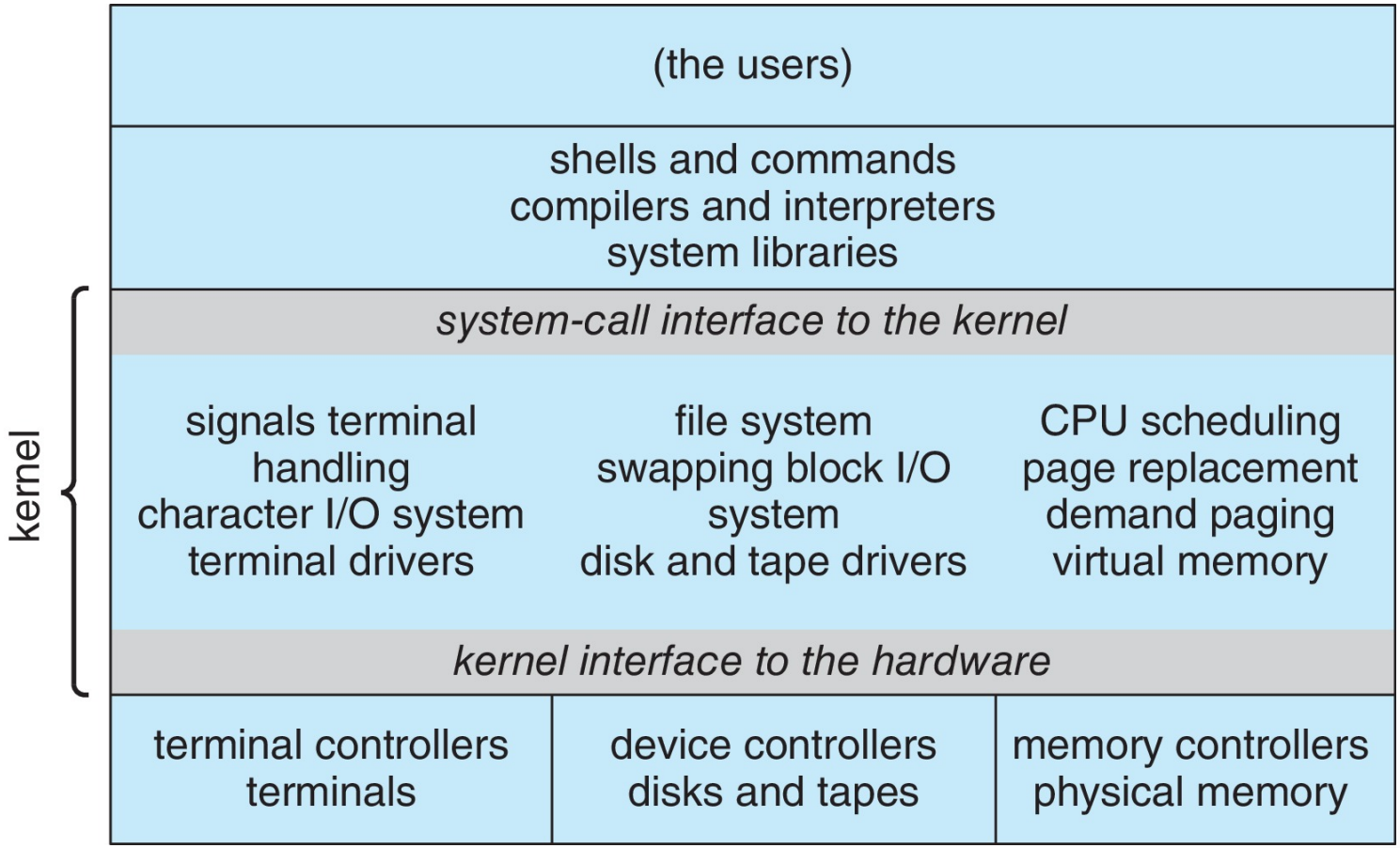
Simple structure – MS-DOS
More complex -- UNIX
Layered – an abstraction
Microkernel -Mach

Simple Structure – MS-DOS



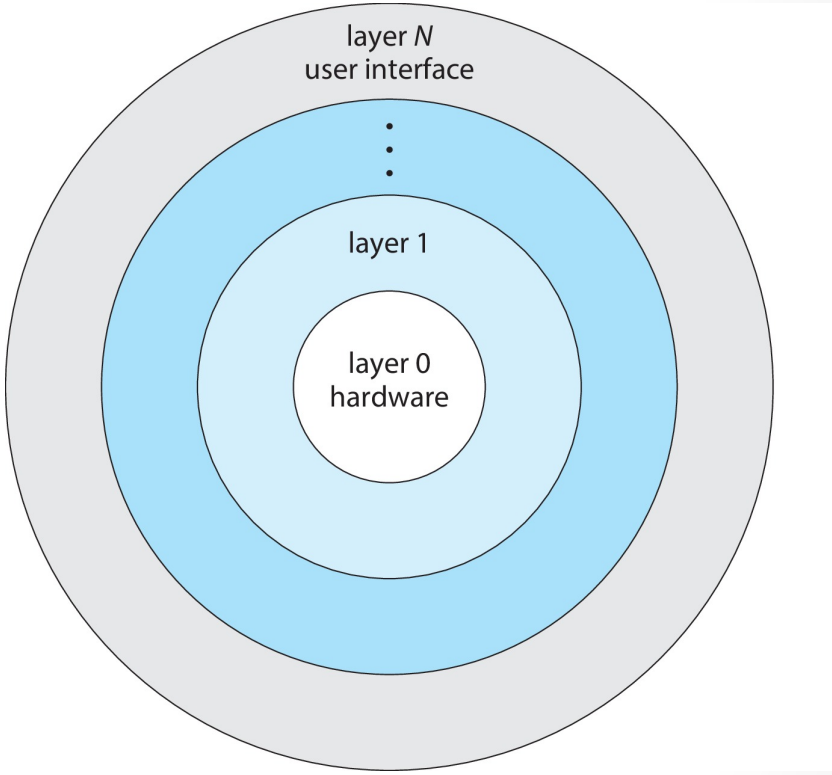
Traditional UNIX System Structure

Beyond simple but not fully layered



Layered Approach

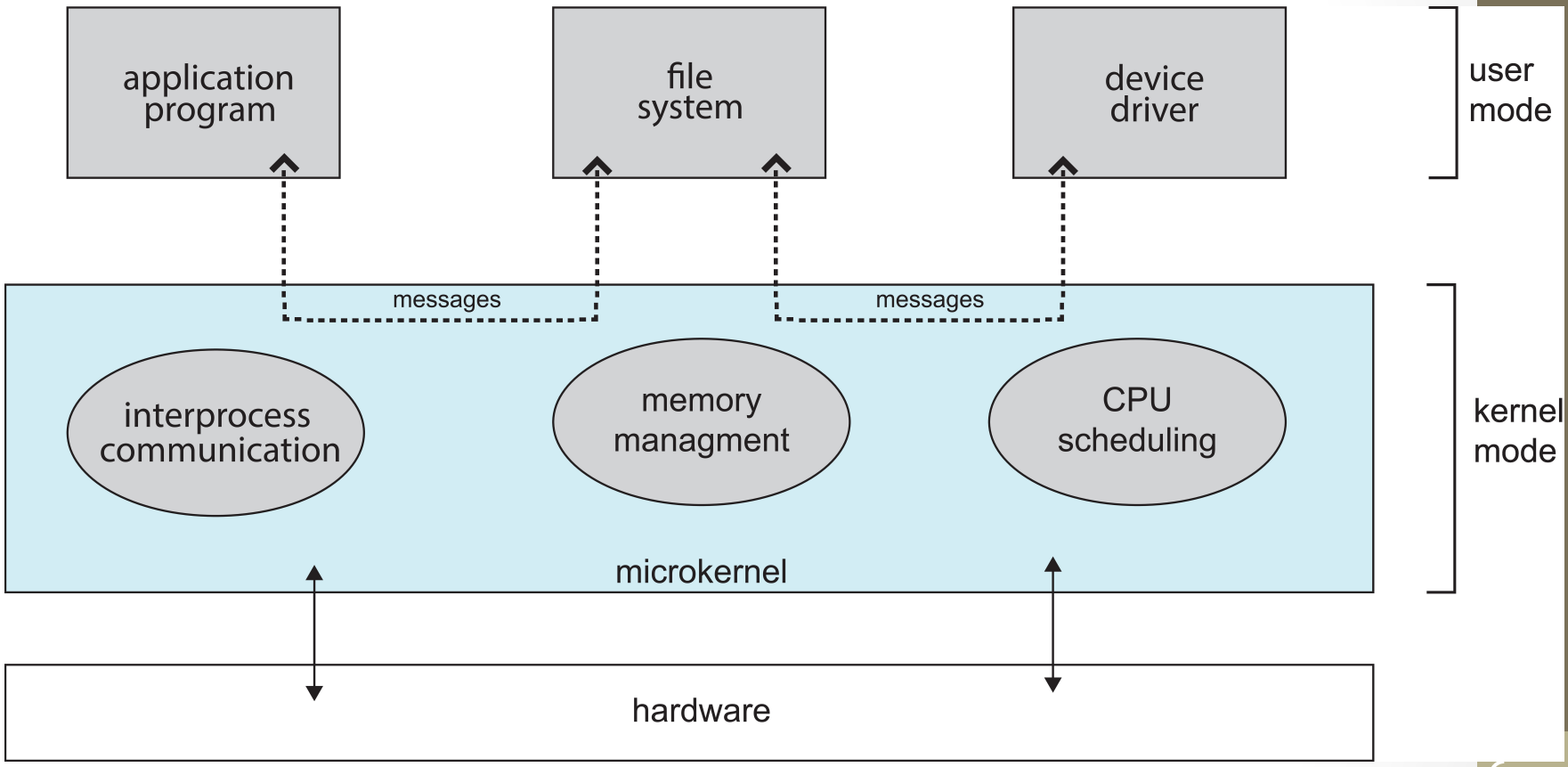
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



Microkernels

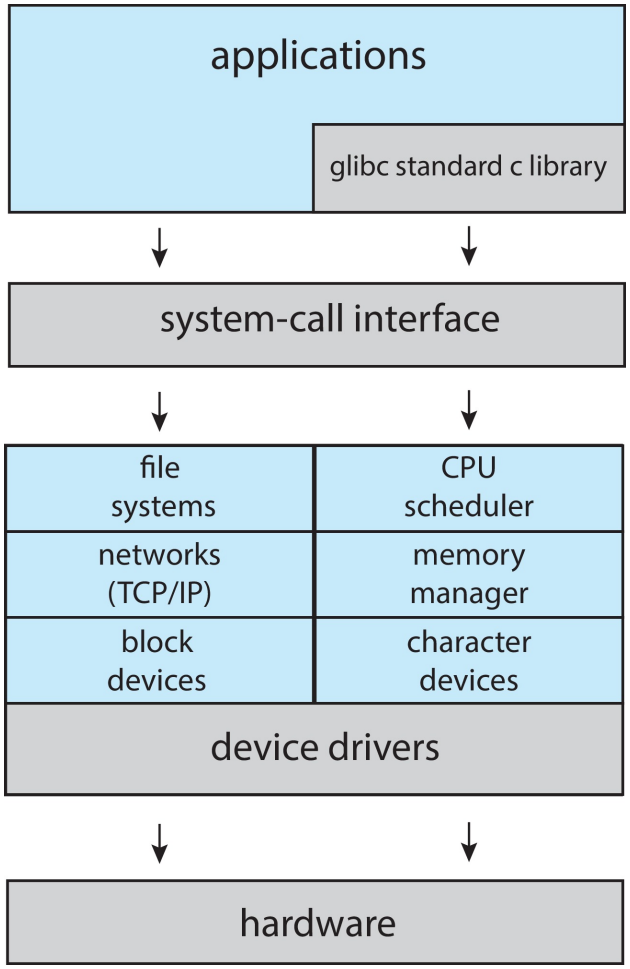
- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Microkernel System Structure



Linux System Structure

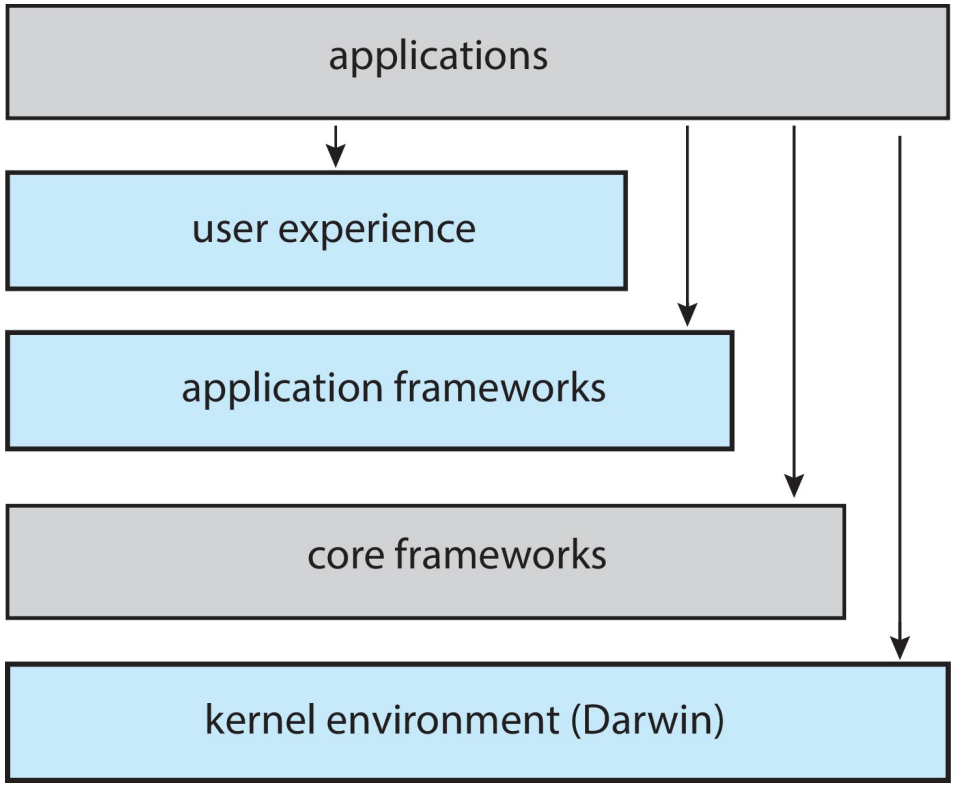
Monolithic plus modular design



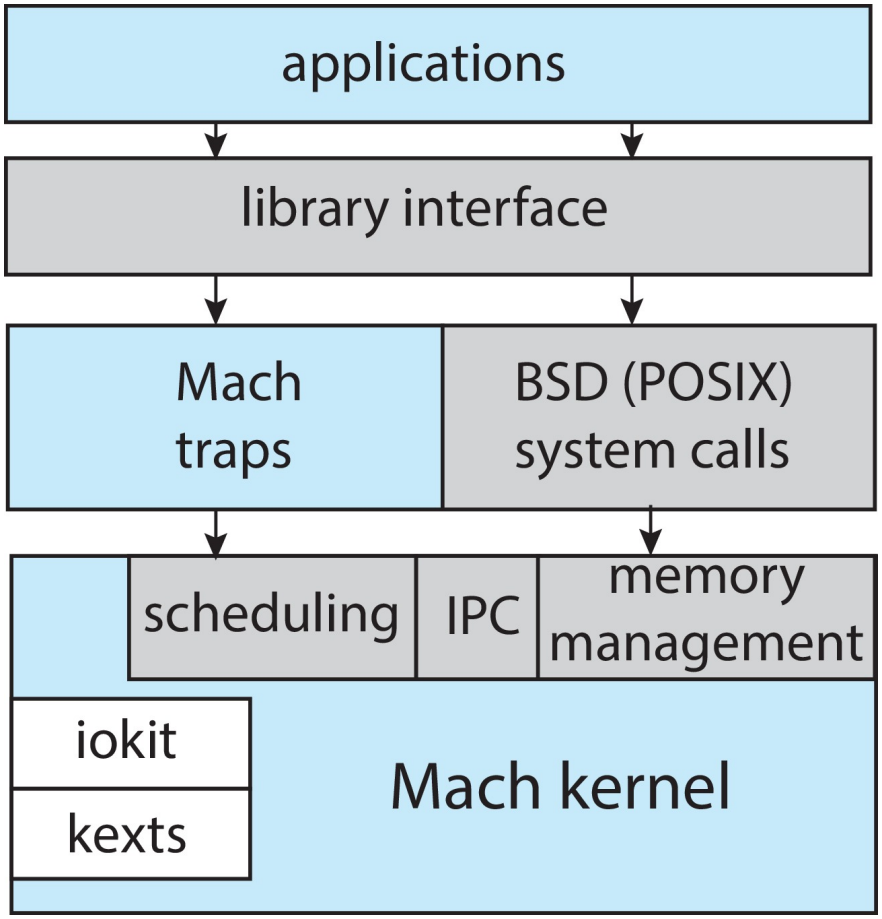
Hybrid Systems

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

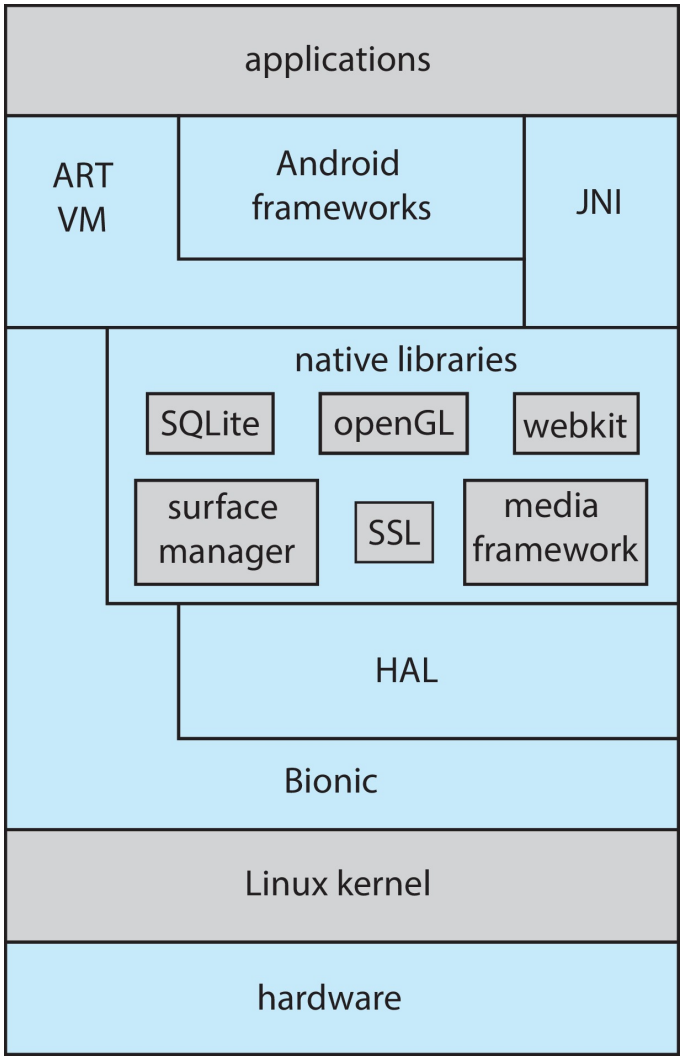
macOS and iOS Structure



Darwin

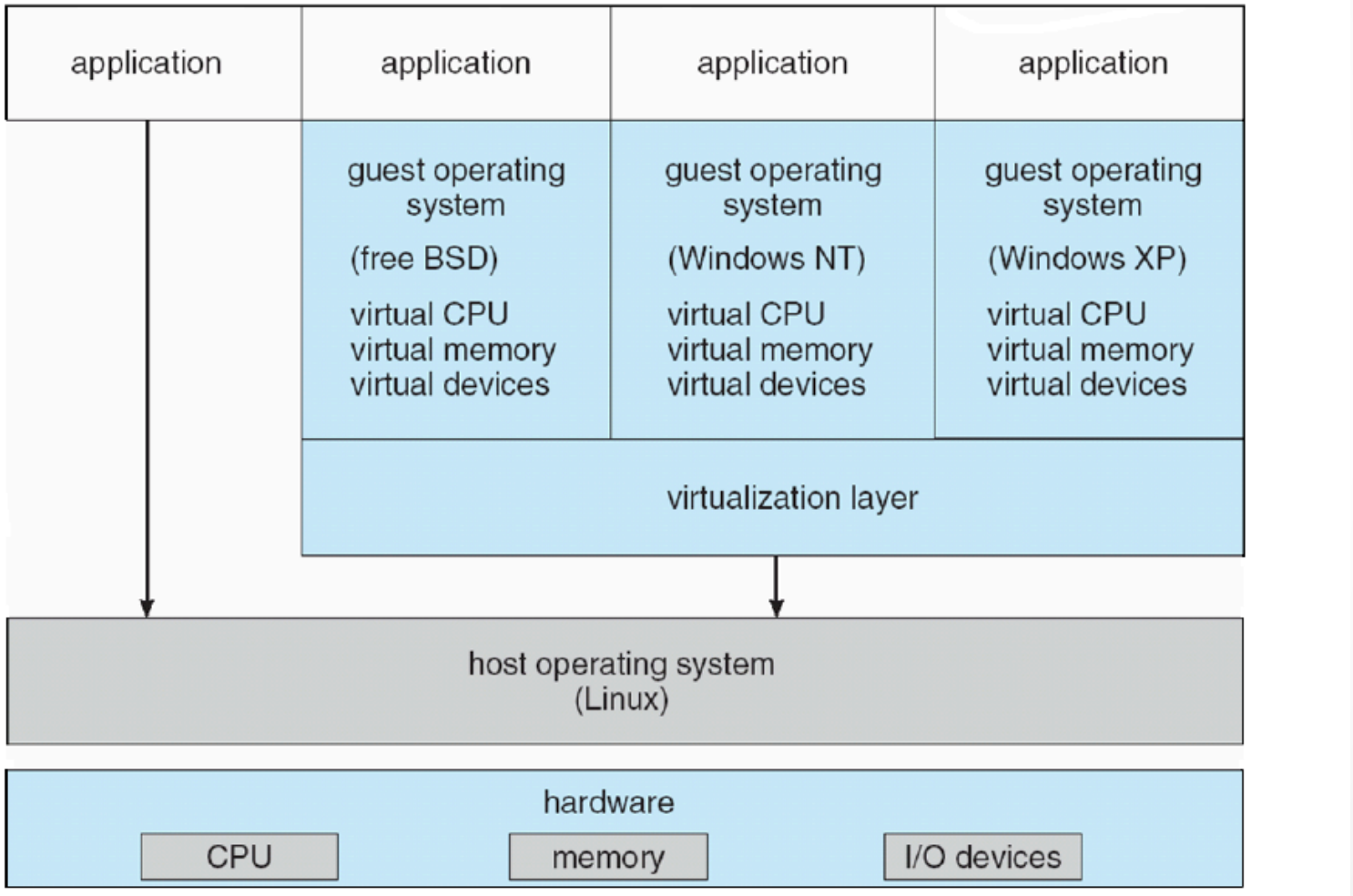

applications
library interface
Mach traps
BSD (POSIX) system calls
scheduling
IPC
memory management
iokit
kexts
Mach kernel

Android Architecture

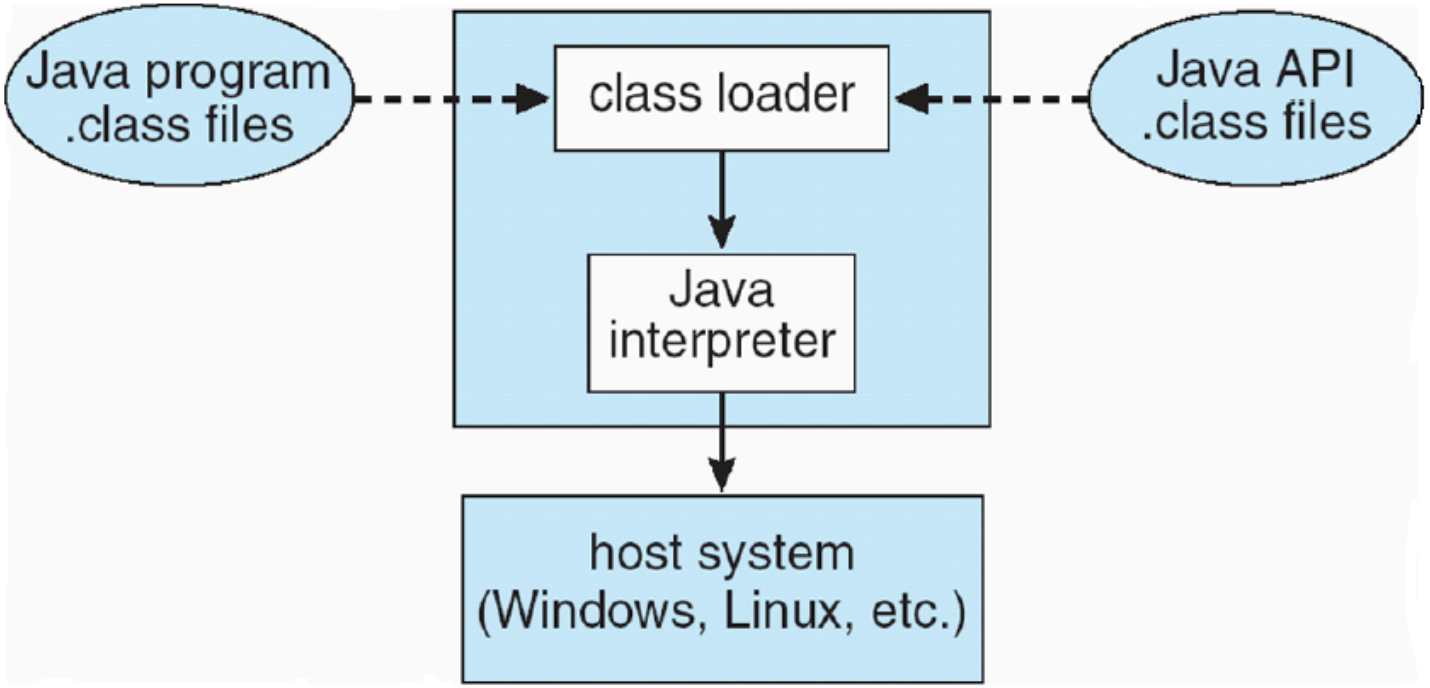


VIRTUAL MACHINE

Vmware architecture



Java Virtual Machine



Homework

- Programming:
  - Creating Kernel Modules, Loading and Removing Kernel Modules, (Chapter 2, Operating System Concept, Silberschatz et al., 9th edition, 2013 page 96-99)
- Revising:
  - Binary, octal, decimal, hex numbers
  - bit, byte, KB, MB, GB, TB, PB, EB, YB, ZB
  - Mbps, Gbps, …

Thank you!

Q&A